Aufgabe 4: Nandu

Team-ID: 00889

Team: Felix Haag

Bearbeiter/-innen dieser Aufgabe:
Felix Haag

12. November 2023

Inhaltsverzeichnis

Lösungsidee....1
Umsetzung....2
Beispiele....3
Quellcode....8

Lösungsidee

Jeden Baustein bekommt zwei Lichtsignale, eines oben und eines unten, die entweder an oder aus sind, als Input und gibt zwei Lichtsignale als Output weiter. Input und Output können somit als boolesche Werte, true für an und false für aus, gesehen werden. Mit der Beschreibung der Funktion des Bausteine, die in der Aufgabe gegeben ist, kann für jeden Baustein der Output abhängig vom Input als boolescher Ausdruck mit den Variablen oben und unten, die jeweils true oder false sein können, dargestellt werden.

Weißer Baustein (Output oben, Output unten): ¬(unten ∧ oben), ¬(unten ∧ oben)

Roter Baustein: ¬oben, ¬oben,

Roter gedrehter Baustein: ¬unten, ¬unten

Blauer Baustein: oben, unten

Um nun den Output eines Aufbaus von Bausteinen zu ermitteln, startet man links in der ganz linken Spalte und ermittelt den Output von jedem Baustein abhängig von den Lichtquellen. Danach fährt man mit der nächsten Spalte fort, diesmal mit den Outputs der Bausteine der vorherigen Spalte als Inputs. So geht mit bis zur letzten Spalte weiter und erhält die endgültigen Outputs.

Um eine Ausgabe in Tabellenform wie das Beispiel in der Aufgabe zu erhalten, muss man dies mit jeder Permutation von true und false für alle Lichtquellen durchführen.

Umsetzung

Vorbereitung

Die Lösungsidee wird in Python implementiert. Zuerst wird für jeden Baustein eine Funktion definiert, der zwei Parameter für die Lichtsensoren links und rechts übergeben werden können und ein Tupel mit zwei Werten für die linke und rechte LED zurückgibt. Diese Funktionen werden in einem Dictionary den Buchstaben W, r, R und B zugeordnet.

Die Eingabedatei wird eingelesen und eine zweidimensionale Liste mit gegebener Höhe und Breite des Aufbaus erstellt und mit None-Werten gefüllt. Danach wird der Aufbau aus der Textdatei in Reihen und an den Leerzeichen in Blöcke gesplittet und mit zwei for-Schleifen durchlaufen. Enthält ein Block ein Q oder L, handelt es sich um eine Lichtquelle oder Ausgabe und diese werden in die Liste an der entsprechenden Position eingetragen. Wird ein anderer Buchstabe erkannt, handelt es sich um einen Baustein und an entsprechender Stelle in der Liste wird der Wert true gespeichert, der dann angibt, dass an dieser und der nächsten Stelle ein Baustein steht. An der nächsten Stelle wird außerdem die dem jeweiligen Buchstaben entsprechende Funktion aus dem Dictionary gespeichert.

Erstellen der Ausgabetabelle

Mit einer for-Schleife wird zuerst das erste Element der Aufbau-Liste durchlaufen, die alle Lichtquellen enthält, um die Anzahl der Lichtquellen zu zählen und diese in den String für die Ausgabetabelle einzutragen. Anschließend wird dasselbe mit dem letzten Element, dass die Ausgabestellen enthält, wiederholt. Danach wird mit einer rekursiven Funktion eine Liste erstellt, die alle Permutation von true und false für die Anzahl an Lichtquellen enthält. Diese Liste wird mit einer for-Schleife durchlaufen und für jede Permutation der Output bestimmt. Anschließend wir die Reihe im Ausgabestring mit den Ein- und Ausgabewerten ergänzt. Zum Schluss wird dieser Ausgabestring in der Konsole ausgegeben und in einer Textdatei gespeichert.

Bestimmen des Outputs

Zum Bestimmen des Outputs für eine Permutation wird zuerst eine zweite zweidimensionale Liste derselben Breite und um eins kleineren Höhe der Liste des Aufbaus erstellt, in der alle Lichtzustände gespeichert werden. Zuerst wird das erste Element der Liste des Aufbaus mit einer for-Schleife durchlaufen und an den Stellen der Lichtquellen die true und false Werte in der Liste für die Lichter eingetragen.

Die Liste des Aufbaus wird jetzt mit zwei for-Schleifen durchlaufen. Hat ein Element den Wert true, so befindet sich an dieser und der nächsten Stelle ein Baustein. Es wird die Funktion des Bausteins, die an der nächsten Stelle gespeichert ist, aufgerufen. Ihr werden als Parameter die Werte aus der Licht-Liste an der derselben Stelle übergeben. Die zurückgegeben Werte werden wiederum in Licht-Liste eine Reihe tiefer gespeichert. So wird der Aufbau von oben nach unten und von links nach rechts durchlaufen, bis in der letzten Reihe der Licht-Liste die Ausgaben der untersten Bausteine gespeichert sind.

Zuletzt wird die letzte Reihe der Aufbau-Liste durchlaufen. An den Stellen, an denen der Output angegeben wird, wird der entsprechende Wert aus der Licht-Liste in einer neuen liste gespeichert. Diese Liste wird zurückgegeben, um die Ergebnisse in den String mit der Ausgabetabelle einzutragen.

Beispiele

nandu0.txt

Q1 Q2 L1 L2

Aus Aus Aus Aus

Aus An Aus Aus

An Aus Aus Aus

An An An An

nandu1.txt

Q1 Q2 L1 L2

Aus Aus An An

Aus An An An

An Aus An An

An An Aus Aus

nandu2.txt

Q1 Q2 L1 L2

Aus Aus Aus An

Aus An Aus An

An Aus Aus An

An An An Aus

nandu3.txt

Q1 Q2 Q3 L1 L2 L3 L4

Aus Aus Aus An Aus Aus An

Aus Aus An An Aus Aus Aus

Aus An Aus An Aus An An

Aus An An An Aus An Aus

An Aus Aus Aus An Aus An

An Aus An Aus An Aus Aus

An An Aus Aus An An An

An An An Aus An An Aus

nandu4.txt

Q1 Q2 Q3 Q4 L1 L2

Aus Aus Aus Aus Aus Aus

Aus Aus Aus An Aus Aus

Aus Aus An Aus Aus An

Aus Aus An An Aus Aus

Aus An Aus Aus An Aus

Aus An Aus An An Aus

Aus An An Aus An An

Aus An An An An Aus

An Aus Aus Aus Aus Aus

An Aus Aus An Aus Aus

An Aus An Aus Aus An

An Aus An An Aus Aus

An An Aus Aus Aus Aus

An An Aus An Aus Aus

An An An Aus Aus An

An An An An Aus Aus

nandu5.txt

Q1	Q2	Q3	Q4	Q5	Q6	L1	L2	L3	L4	L5
Aus	Aus	Aus	Aus	Aus	Aus	Aus	Aus	Aus	An	Aus
Aus	Aus	Aus	Aus	Aus	An	Aus	Aus	Aus	An	Aus
Aus	Aus	Aus	Aus	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	Aus	Aus	An	An	Aus	Aus	Aus	An	An
Aus	Aus	Aus	An	Aus	Aus	Aus	Aus	An	Aus	Aus
Aus	Aus	Aus	An	Aus	An	Aus	Aus	An	Aus	Aus
Aus	Aus	Aus	An	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	Aus	An	An	An	Aus	Aus	Aus	An	An
Aus	Aus	An	Aus	Aus	Aus	Aus	Aus	Aus	An	Aus
Aus	Aus	An	Aus	Aus	An	Aus	Aus	Aus	An	Aus
Aus	Aus	An	Aus	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	An	Aus	An	An	Aus	Aus	Aus	An	An
Aus	Aus	An	An	Aus	Aus	Aus	Aus	An	Aus	Aus
Aus	Aus	An	An	Aus	An	Aus	Aus	An	Aus	Aus
Aus	Aus	An	An	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	An	An	An	An	Aus	Aus	Aus	An	An
Aus	An	Aus	Aus	Aus	Aus	Aus	Aus	Aus	An	Aus
Aus	An	Aus	Aus	Aus	An	Aus	Aus	Aus	An	Aus
Aus	An	Aus	Aus	An	Aus	Aus	Aus	Aus	An	An
Aus	An	Aus	Aus	An	An	Aus	Aus	Aus	An	An
Aus	An	Aus	An	Aus	Aus	Aus	Aus	An	Aus	Aus
Aus	An	Aus	An	Aus	An	Aus	Aus	An	Aus	Aus
Aus	An	Aus	An	An	Aus	Aus	Aus	Aus	An	An
Aus	An	Aus	An	An	An	Aus	Aus	Aus	An	An
Aus	An	An	Aus	Aus	Aus	Aus	Aus	Aus	An	Aus
Aus	An	An	Aus	Aus	An	Aus	Aus	Aus	An	Aus
Aus	An	An	Aus	An	Aus	Aus	Aus	Aus	An	An

Aus An An Aus An An Aus Aus Aus An An

Aus An An An Aus Aus Aus Aus An Aus Aus

Aus An An An Aus An Aus Aus An Aus Aus

Aus An An An An Aus Aus Aus Aus An An

Aus An An An An An Aus Aus Aus An An

An Aus Aus Aus Aus Aus An Aus Aus An Aus

An Aus Aus Aus Aus An An Aus Aus An Aus

An Aus Aus Aus An Aus An Aus Aus An An

An Aus Aus Aus An An An Aus Aus An An

An Aus Aus An Aus Aus An Aus An Aus Aus

An Aus Aus An Aus An An Aus An Aus Aus

An Aus Aus An An Aus An Aus Aus An An

An Aus Aus An An An An Aus Aus An An

An Aus An Aus Aus Aus An Aus Aus An Aus

An Aus An Aus Aus An An Aus Aus An Aus

An Aus An Aus An Aus An Aus Aus An An

An Aus An Aus An An An Aus Aus An An

An Aus An An Aus Aus An Aus An Aus Aus

An Aus An An Aus An An Aus An Aus Aus

An Aus An An An Aus An Aus Aus An An

An Aus An An An An An Aus Aus An An

An An Aus Aus Aus Aus An Aus Aus An Aus

An An Aus Aus Aus An An Aus Aus An Aus

An An Aus Aus An Aus An Aus Aus An An

An An Aus Aus An An An Aus Aus An An

An An Aus An Aus Aus An Aus An Aus Aus

An An Aus An Aus An An Aus An Aus Aus

An An Aus An An Aus An Aus Aus An An

An An Aus An An An An Aus Aus An An

An An An Aus Aus Aus An Aus Aus An Aus

An An An Aus Aus An An Aus Aus An Aus

An An An Aus An Aus An Aus Aus An An

An An An Aus An An An Aus Aus An An

An An An An Aus Aus An Aus An Aus Aus

An An An An Aus An An Aus An Aus Aus

An An An An An Aus An Aus Aus An An

An An An An An An An Aus Aus An An

Quellcode

Einlesen der Textdatei

```
def read_file(filename):
    dispatcher = {"W": white, "r": red, "R": red_turned, "B": blue}
    with open(os.path.dirname(__file__) + "/" + filename, "r") as f:

        dims = f.readline().split(" ")
        width = int(dims[0].replace("\n", ""))
        height = int(dims[1].replace("\n", ""))

       # Zweidimensionale Liste für Aufbau der Bausteine, None = kein Baustein
        structure = [[None for _ in range(0, width)] for _ in range(0, height)]

        lines = f.readlines() # Zeilen der Eingabe
        split_lines = [] # Einzelne Zeichen der Eingabe

        # Aufsplitten der Zeilen der Eingabe
        for n, line in enumerate(lines):
            split_lines.append([])
            for char in line.split(" "):
                if char:
                    split_lines[n].append(char)

        skip = [] # Position die überspungen werden sollen

        # Durchlaufen des Aufbaus
        for i in range(0, height):
            for j in range(0, width):

                if (i, j) in skip:
                    continue

                block = split_lines[i][j].strip().replace("\n", "") # Zeichen an der gegebenen Position

                # Falls an der Stelle eine Lichtquelle ist
                if "Q" in block:
                    structure[i][j] = block
                # Falls an der Stelle eine Ausgabe ist
                elif "L" in block:
                    structure[i][j] = block
                # Falls die Stelle nicht leer ist
                elif block != "X":
                    # True gibt an das an Position (i, j) und (i, j+1) ein Baustein ist
                    structure[i][j] = True
                    structure[i][j+1] = dispatcher[block] # Speichern der Funktion je nach Baustein
                    skip.append((i, j+1)) # Position (i, j+1) muss nicht nochmal untersucht werden

    return structure, width, height
```

Baustein Funktionen

```
def white(left, right): # weißer Baustein
    on = not(left and right) # ¬(right ∧ ¬right)
    return (on, on)

def red(_, right): # roter Baustein mit Lichtsensor oben
    return (not(right), not(right)) # ¬right, ¬right

def red_turned(left, _): # roter Baustein gedreht mit Lichtsensor unten
    return (not(left), not(left)) # ¬left, ¬left

def blue(left, right): # blauer Baustein
    return (left, right) # left, right
```

Boolean Permutationen

```
# Erstellen einer Liste mit allen Permutation von True und False, bei n Lichtquellen
def bool_permutations(n, l, i_c):
    if n == 0:
        i_c.append(l)
    else:
        return [bool_permutations(n-1, l+[True], i_c)] + [bool_permutations(n-1, l+[False], i_c)]
```

Output für eine Permutation

```
def get_output(comb, structure, width, height):
    # Speichern der Lichter, False = Licht aus
    light = [[False for _ in range(width)] for _ in range(height-1)]

    # Einsetzen der True und False Werte an den Stellen der Lichtquellen
    i = 0
    for n, block in enumerate(structure[0]):
        if block:
            light[0][n] = comb[i]
            i += 1

    # Durhclaufen des Aufbaus von oben nach unten und links nach rechts
    for i, row in enumerate(structure[1:]):
        for j, block in enumerate(row):

            # Auf diesem und dem nächsten Feld befindet sich ein Baustein
            if block == True:
                # Bestimmen der Ausgabe des Bausteins, Bsp: (true, true) = white(true, false)
                erg = structure[i+1][j+1](light[i][j], light[i][j+1])
                light[i+1][j] = erg[0] # Speichern der Ausgabe in Lichter-Liste
                light[i+1][j+1] = erg[1]

    # Speichern der Ausgaben an den gekennzeichneten Stellen (L1, L2, ...)
    output = []
    for n, block in enumerate(structure[-1]):
        if block:
            output.append(light[-1][n])

    return output
```

Erstellen des Ausgabetabelle

```
def create_output_string(structure, width, height):
    output_string = ""
    inputs = 0
    outputs = []

    # Zählen der Lichtquellen
    for block in structure[0]:
        if block:
            inputs += 1

            # Hinzufügen der Lichtquelle zum Output
            output_string += block + "    "

    for block in structure[-1]:
        if block:
            # Hinzufügen der Ausgaben zum Output
            output_string += block + "    "

    output_string += "\n"

    # Erstellen aller Permutation von True und False, bei Anzahl der Lichtquellen
    input_combinations = []
    bool_permutations(inputs, [], input_combinations)

    # Durchlaufen aller Permutationen
    for comb in reversed(input_combinations):
        for bool in comb:
            # Wert der Lichtquelle (True = An, False = Aus) zur Output hinzufügen
            output_string += "An    " if bool else "Aus   "

        # Ermitteln der Ausgaben der unteren Bausteine
        outputs = get_output(comb, structure, width, height)

        # Hinzufügen der Ergebnisse zum Output
        for bool in outputs:
            output_string += "An    " if bool else "Aus   "

        output_string += "\n"

    return output_string
```